EXHIBIT A

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF MASSACHUSETTS

SINGULAR COMPUTING LLC, Plaintiff, v. GOOGLE LLC, Defendant.	C.A. No. 1:19-cv-12551-FDS

EXPERT REPORT OF MIRIAM LEESER, PH.D. REGARDING INVALIDITY

Executed on 12/20/2022

DocuSigned by:
Miriam Leeser, Ph.D.
A3ED90A1A36A4F5...

Miriam Leeser, Ph.D.

Technology's (MIT) Lincoln Laboratories, which is a federally funded research and development center affiliated with MIT and operated by the U.S. Department of Defense, and which was chartered to apply advanced technology to problems of national security.

16. I am an author or co-author of over 180 refereed publications, including 45 peer-reviewed published journal articles, as well as the book *Hardware Specification, Verification, and Synthesis: Mathematical Aspects*. I have been an invited speaker at over 70 conferences, symposia, and workshops on topics within my area of research and expertise. I have received extensive recognition for my published research in the areas of high performance computing and reconfigurable hardware, including various best paper awards; for example, for my paper "Accelerating Matrix Processing for MIMO Systems" with Jieming Xu at the International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 2021), and my paper "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," which was published in ACM Transactions on Reconfigurable Technology and Systems (TRETS), Special Section on Deep Learning on FPGAs, Vol. 11, Issue 3 (December 2018), and received a best paper award from ACM TRETS in 2020.

17. I am also a named inventor on U.S. Patent No. 4,967,344, which is titled "Interconnection Network for Multiple Processors" and generally relates to techniques for connecting a large number of independent processors together using a packet-based network (data bus) to allow very high-throughput data transfer between the processors. This patent arose from my time working in industry (at Codex Corp.).

18. Additional information regarding my background, qualifications, and experience, including a list of publications, conference and symposium presentations, and talks, can be found in my *curriculum vitae*, which is attached as **Exhibit B**.

III. SUMMARY OF OPINIONS

19. For the reasons set forth in the body of this report, it is my opinion that claim 53 of the '273 patent is invalid as anticipated due to the claimed invention being known or used by

others in this country before the priority date of the Asserted Patents, being in public use in this country more than one year prior to the priority date for the Asserted Patents, and/or being previously made in this country by another inventor who had not abandoned, suppressed, or concealed the invention, based on the creation, public availability, public disclosure, public knowledge, and public use of and regarding VFLOAT and its use on FPGA hardware. In addition, it is my further opinion that, even if claim 53 of the '273 patent is not anticipated, the claimed invention would have been obvious to a person having ordinary skill in the art at the time based on the creation, public availability, public disclosure, public knowledge, and public use of and regarding VFLOAT in combination with FPGA hardware available at the time of the claimed invention.

20. Likewise, it is my opinion that claim 7 of the '156 patent is invalid as anticipated due to the claimed invention being known or used by others in this country before the priority date of the Asserted Patents, being in public use in this country more than one year prior to the priority date for the Asserted Patents, and/or being previously made in this country by another inventor who had not abandoned, suppressed, or concealed the invention, based on the creation, public availability, public disclosure, public knowledge, and public use of and regarding VFLOAT and its use on FPGA hardware. In addition, it is my further opinion that, even if claim 53 of the '273 patent is not anticipated, the claimed invention would have been obvious to a person having ordinary skill in the art at the time based on the creation, public availability, public disclosure, public knowledge, and public use of and regarding VFLOAT in combination with FPGA hardware available at the time of the claimed invention

IV. APPLICABLE LEGAL STANDARDS

21. I am not an attorney and have no formal legal training. I have been informed of the relevant legal standards that apply in evaluating the validity of a patent from Google's attorneys and am relying on those instructions for these legal standards. Below I describe my understanding of these legal standards.

reconfigurable computing engine, WILDSTAR 5, featured 3 Xilinx Virtex-5 FPGAs, such as three Xilinx Virtex-5 XC5VLX330T FPGAs.[25]

73. In addition to the physical WILDSTAR board itself, Annapolis Micro provided software referred to as a board support package (BSP) that ran on the x86 host machine. The BSP supported downloading the bitstream from the host, runtime communication with the host, and communication with on-board memory.

VI. THE VFLOAT LIBRARY

74. In this section, I provide an overview of the VFLOAT library, including what VFLOAT was, the origins or motivation that originally led to the development of VFLOAT, who was involved in the work, our deployment of VFLOAT on reconfigurable computing hardware, and the information that I and others made publicly available about VFLOAT and related work.

A. VFLOAT Overview / Background

75. VFLOAT, also known more descriptively as "The Northeastern Variable-Precision Floating Point Library," was a library (or set) of parameterized hardware modules for performing arithmetic operations on floating-point numbers of various formats using reconfigurable hardware, namely FPGAs. What that means is that a designer using VFLOAT could quickly deploy, onto FPGA hardware, circuits containing multiplication units and other floating-point arithmetic modules that operate on a variety of customizable floating-point number formats. The designer simply selects the floating-point number format they wish to use by specifying bitwidths for the exponent and fraction; the code in the library could then create the corresponding VFLOAT arithmetic modules for implementation on FPGAs.

76. Our general motivation in developing VFLOAT was the idea or understanding that, when implementing floating-point arithmetic operators on reconfigurable hardware such as FPGAs, fine-grained control over the floating-point formats used is possible and even desirable because the optimal bitwidth of a signal often depends on the particular application, and the

[25] *See* https://web.archive.org/web/20081004190813/http://www.annapmicro.com/ws5pci.html [LEESER000120].

further idea or understanding that flexible control over bitwidths allows greater parallelism and more efficient use of hardware resources.

77. VFLOAT was developed by myself and others in the 2000-2002 timeframe at the Reconfigurable Computing Laboratory (RCL) at Northeastern University (back when it was known as the Rapid Prototyping Laboratory or RPL), the group that I have led since 1996, including during the 2000-20002 timeframe when VFLOAT was first developed. Specifically, in addition to myself, other contributors to the original VFLOAT library included Pavle Belanović, who at the time was a graduate student in Electrical & Computer Engineering at Northeastern University whose graduate work I supervised. Several other graduate students whom I supervised also contributed to subsequent iterations or revisions of VFLOAT over the years, including Haiqian Yu in 2003 and Xiaojun Wang in 2008, although the focus of this report is the original version of VFLOAT that was developed and made publicly available in approximately 2002.

78. The origins or genesis of VFLOAT can be traced to my work for Los Alamos National Laboratory (LANL), in approximately 1999-2002. LANL, which is located just outside Santa Fe, New Mexico, is one of a few national laboratories operated by the U.S. Department of Energy and is most well known for first being organized during World War II for the design of nuclear weapons as part of the Manhattan Project. I began working for LANL in approximately 1999 on a sub-contract received by Northeastern University, which was funded by a grant from the U.S. Defense Advanced Research Projects Agency (DARPA). DARPA is the agency of the U.S. Department of Defense responsible for researching and developing emerging technologies for use by the military, though those technologies often end up having significant application in the commercial or private sector. The Internet is probably the most well-known example of a technology originally developed by DARPA; its precursor was DARPA's ARPANET, the first wide-area packet-switched computer network.

79. The formal title of the sub-contract with LANL was "Acceleration of Scene Classification and Spectral Unmixing with Reconfigurable Computing," and is identified on my

CV under "Research Grants and Contracts." In general, my objective for this sub-contract with LANL was to explore ways to use FPGAs to quickly analyze large volumes of satellite data that LANL received and needed to process, and more specifically to accelerate their ability to analyze this data compared to existing techniques. This project for LANL supported both myself and several of my students at the time, including Mr. Belanović, who (as mentioned earlier) was a graduate student pursuing his Masters of Science degree in Electrical Engineering at Northeastern University. The individuals we worked and interfaced with at LANL were officials in the Space and Remote Sensing Sciences Group.

80. VFLOAT became the deliverable for our work for LANL. Under my guidance, Mr. Belanović developed a library of hardware modules for performing variable-precision floating-point arithmetic on FPGAs, which we later named VFLOAT—the "FLOAT" signifying that the modules in the library were designed to perform arithmetic operations using floating-point numbers and the "V" signifying that the modules were *variable* in that they used custom, user-defined floating-point number formats (that is, floating-point formats that could have different numbers of exponent and fraction bits as compared to the IEEE standard single-precision format).

81. The initial version of VFLOAT was developed specifically for our work for LANL, and LANL directed us to open-source (*i.e.*, non-classified) data sets to validate the use of VFLOAT for their specific application (analyzing satellite imagery and other data). That is also the reason why a number of our presentations and papers related to VFLOAT over the years identify satellite data processing as an example application of VFLOAT. For example, in my presentation at HPEC 2002 (discussed in more detail below), I presented on, among other things, a hybrid implementation of the K-means clustering algorithm using VFLOAT in the specific context of satellite imagery. This is reflected in the slides I used for that HPEC 2002 presentation. *See* **Exhibit C**.

82. At an earlier stage in our work for LANL, we had developed an implementation of K-means clustering[26] in reconfigurable hardware (FPGAs) called "ASAPP" that proved to be orders of magnitude faster than implementing the same algorithm in software. But because it was designed for one dataset with a particular set of parameters (multispectral images with 10 channels, 12 bits per channel, and 8 clusters, to match MTI data[27]), we subsequently developed a parameterized hardware implementation of K-means clustering to more easily adapt to various different datasets available within the domain of multispectral or hyperspectral satellite imagery (or other domains)—for example, datasets with a different number of channels, with a different number of bits per channel, with a different number of pixels, and/or requiring a different number of clusters.[28] VFLOAT was very much a logical extension of this work in that it carried the concept of parameterization into the underlying floating-point arithmetic modules used to implement K-means clustering.

83. As discussed in more detail below, I personally traveled to New Mexico several times during the course of our work with the Space and Remote Sensing Sciences Group to discuss and present our work related to VFLOAT to the officials there, and Mr. Belanović accompanied me on at least one occasion. Based on my interactions with the Space and Remote Sensing Sciences Group, I believe that LANL ultimately deployed VFLOAT for the purpose of analyzing satellite imagery and other data using actual FPGA hardware.

[26] K-means clustering is a common technique for segmentation of multi-dimensional data, such as multispectral and hyperspectral satellite imagery.

[27] MTI refers to the "Multispectral Thermal Imager," which was a Department of Energy-funded satellite in polar orbit that was launched in early 2000 and carried an advanced multispectral and thermal imaging sensor.

[28] This work on a parameterized hardware implementation of K-means clustering is described in more detail in "Applying Reconfigurable Hardware to the Analysis of Multispectral and Hyperspectral Imagery" , a paper that I co-authored along with Mr. Belanović, Michael Estlick, and three individuals from LANL with whom we worked (Maya Gokhale, John Szymanski, and James Theiler), and which I presented at the 2001 International Symposium on Optical Science and Technology. *See Proceedings - SPIE*, Vol. 4480 (2001), https://www.spiedigitallibrary.org/conference-proceedings-of-spie/4480/0000/Applying-reconfigurable-hardware-to-the-analysis-of-multispectral-and-hyperspectral/10.1117/12.453329.short [LEESER000122].

B. Detailed Discussion of VFLOAT

84. As described above, FPGAs are a form of reconfigurable hardware whose structure or architecture can be specified by a designer after manufacturing, typically using a Hardware Description Language (HDL). In the case of VFLOAT, we programmed the design of the various hardware modules in VHDL (VHSIC Hardware Description Language). When mapped to FPGA hardware, VFLOAT modules define circuits that are structured and behave as specified by the VHDL code. The full set of code comprising the original VFLOAT library is attached to my report as **Exhibit D**.

1. Hardware Modules Comprising VFLOAT

85. At a high level, VFLOAT comprises parameterized floating-point arithmetic modules, defined in VHDL, for performing floating-point addition (*fp_add*[29]) and floating-point subtraction (*fp_sub*[30]), as well as a module that performs the exponent addition and mantissa multiplication steps used in floating-point multiplication (*fp_mul*[31]). VFLOAT also comprises a number of low-level hardware modules that are used as building blocks in these top-level arithmetic operators, such as a fixed-point multiplier (*parameterized_multiplier*[32]), which is needed, for example, to perform mantissa multiplication. VFLOAT also comprises a number of format control modules, such as a denormalization module (*denorm*[33]), which adds back the implied '1' to the fraction field to form a mantissa with a non-zero integer part, and a rounding and normalization module (*rnd_norm*[34]), which performs rounding and normalization of floating-point values created by the arithmetic modules. VFLOAT further comprises various other categories of modules, such as parameterized modules for converting from fixed-point to floating-point number formats (*fix2float*[35]) and vice versa (*float2fix*[36]).

[29] *See* Exhibit D at fp_add.vhd.
[30] *See* Exhibit D at fp_sub.vhd.
[31] *See* Exhibit D at fp_mul.vhd.
[32] *See* Exhibit D at parameterized_multiplier.vhd.
[33] *See* Exhibit D at denorm.vhd.
[34] *See* Exhibit D at rnd_norm.vhd.
[35] *See* Exhibit D at fix2float.vhd.
[36] *See* Exhibit D at float2fix.vhd.

86. As detailed below, the arithmetic modules comprising VFLOAT accept exponents that are biased using a biasing scheme similar to that used in the IEEE single-precision format.

2. Parameterization of VFLOAT Hardware Modules

87. As noted above, the floating-point arithmetic modules comprising VFLOAT are parameterized, which means that they could operate on essentially arbitrary floating-point number formats selected by the designer. In fact, nearly all the modules (including the low-level and format-control modules discussed above) that comprise the VFLOAT library are parameterized. This parameterization can be seen directly in the source code for VFLOAT.

88. For example, the module *fp_mul* (which, as noted above, is the module that performs the exponent addition and mantissa multiplication steps used in floating-point multiplication) has two floating-point number inputs, `OP1` and `OP2`.[37] The VHDL `port` declaration[38] for *fp_mul* explicitly defines those inputs (`OP1` and `OP2`) based on two parameters: the bitwidth of the exponent for the selected floating-point format (`exp_bits`) and the bitwidth of the mantissa for the selected floating-point format (`man_bits`). Specifically, the inputs `OP1` and `OP2` are each defined as *n*-element vectors, using the `std_logic_vector` data type, where *n* = `exp_bits` + `man_bits` + 1).[39] These parameters (`exp_bits` and `man_bits`), in turn, are defined in the `generic` clause of *fp_mul*, where they can be customized in accordance with the designer's specific application by specifying the exponent bitwidth and mantissa bitwidth.[40] When instantiated, the *fp_mul* module creates a circuit on the FPGA that accepts floating-point inputs with the exponent and mantissa bitwidths specified by `exp_bits` and `man_bits`.

[37] *See* Exhibit D at fp_mul.vhd:65-77.
[38] "In a VHDL Output File (. vho), a port name in the Entity Declaration represents an input or output of the current file. When an instance of a primitive or lower-level design file is implemented with a Component Instantiation, its ports are connected to signals with Port Map Aspects." *See* https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/reference/glossary/def_port.htm#:~:text=In%20a%20VHDL%20Output%20File,signals%20with%20Port%20Map%20Aspects [LEESER000157].
[39] *See* Exhibit D at fp_mul.vhd:68-69.
[40] *See* Exhibit D at fp_mul.vhd:60-64.

89. Similarly, the module *denorm* (which, as noted above, is the module that performs denormalization of floating-point values) has a floating-point input, `IN1`, on which it performs the denormalization operation, and a floating-point output, `OUT1`, to output the results of the denormalization.[41] The VHDL `port` declaration for *denorm* explicitly defines the input (`IN1`) and output (`OUT1`) based on two parameters: the bitwidth of the exponent for the selected floating-point format (`exp_bits`) and the bitwidth of the mantissa for the selected floating-point format (`man_bits`). Specifically, input `IN1` is defined as an *n*-element vector using the `std_logic_vector` data type, where *n* = `exp_bits` + `man_bits` + 1 (the additional bit beyond the sum of parameters `exp_bits` and `man_bits` is for the sign bit[42]) and output `OUT1` is defined as an *n*+1-element vector (to accommodate the implied '1' added to create the mantissa).[43] These parameters (`exp_bits` and `man_bits`), in turn, are defined in the `generic` clause of *denorm*, where they can be customized in accordance with the designer's specific application by specifying the exponent bitwidth and mantissa bitwidth.[44] When instantiated, the *denorm* module creates a circuit on the FPGA that accepts a floating-point input with the exponent and mantissa bitwidths specified by `exp_bits` and `man_bits` and adds back the implied '1' to form a mantissa with a non-zero integer part.

90. Likewise, the module *rnd_norm* (which, as noted above, is the module that performs rounding and normalization of floating-point values) has a floating-point input, `IN1`, on which it performs normalization and rounding, and a floating-point output, `OUT1`, to output the results of the normalization and rounding.[45] The VHDL `port` declaration for *rnd_norm* explicitly defines the input (`IN1`) based on two parameters: the bitwidth of the exponent for the selected floating-point format (`exp_bits`) and the bitwidth of the input mantissa (`man_bits_in`). Specifically, the input `IN1` is defined as an *n*-element vector using the

[41] *See* Exhibit D at denorm.vhd:60-70.
[42] *See* Exhibit D at denorm.vhd:87 (connecting sign signal to most-significant bit of `IN1: s_old <= IN1(man_bits+exp_bits)`).
[43] *See* Exhibit D at denorm.vhd:63, 67.
[44] *See* Exhibit D at denorm.vhd:55-59.
[45] *See* Exhibit D at rnd_norm.vhd:61-73.

`std_logic_vector` data type, where *n* = `exp_bits` + `man_bits_in` + 1.[46] Likewise, the VHDL `port` declaration for *rnd_norm* explicitly defines the output (`OUT1`) based on two parameters: the bitwidth of the exponent for the selected floating-point format (`exp_bits`) and the bitwidth of the output mantissa (`man_bits_out`). Specifically, the output `OUT1` is defined as an *m*-element vector using the `std_logic_vector` data type, where *m* = `exp_bits` + `man_bits_out` + 1.[47] These parameters (`exp_bits`, `man_bits_in`, and `man_bits_out`), in turn, are defined in the `generic` clause of *rnd_norm*, where they can be customized in accordance with the designer's specific application by specifying the exponent bitwidth and mantissa bitwidths for the input and output.[48] When instantiated, the *rnd_norm* module creates a circuit on the FPGA that accepts a floating-point input with the exponent and mantissa bitwidths specified by `exp_bits` and `man_bits_in` and outputs a rounded and normalized floating-point value (*i.e.*, the floating-point input after rounding and normalization) with the exponent and mantissa bitwidths specified by `exp_bits` and `man_bits_out`.

3. VFLOAT Multiplier

91. As noted, the VFLOAT library was comprised of parameterizable modules that were designed to be—and were—used in combination, kind of like Lego building blocks for programming FPGA circuits. For example, 2 *denorm*, 1 *fp_mul*, and 1 *rnd_norm* would be assembled to create a complete multiplier circuit.[49]

92. The size of the various floating-point hardware modules created on FPGAs using the VFLOAT library, including *denorm*, *fp_mul*, and *rnd_norm*, was dependent on the bitwidth of the floating-point numbers those modules operated on. In the multiplier hardware created on FPGAs by the VFLOAT library, the overall size of a floating-point multiplier is dominated by the hardware needed to multiply the mantissas of two numbers; thus, for a given total bitwidth,

[46] *See* Exhibit D at rnd_norm.vhd:64.
[47] *See* Exhibit D at rnd_norm.vhd:70.
[48] *See* Exhibit D at rnd_norm.vhd:55-60
[49] *See* Exhibit C at 11 (HPEC 2002 Slides) (describing assembly of modules for an IEEE single-precision adder).

allocating fewer bits to the fraction field and more to the exponent field reduces the overall size of the multiplier.

a. VFLOAT Floating-Point Denormalization Module: *denorm*

93. As noted above, the parameterized module in VFLOAT for performing denormalization on floating-point values (*i.e.*, adding the implied '1' to the fraction field to form a mantissa with a non-zero integer part) was named *denorm*, and its design (*i.e.*, structure and behavior) is expressed in the VHDL source code specified in denorm.vhd.[50] The *denorm* module accepts as input one floating-point number and outputs the denormalized form of that number. As can be seen in the source code for *denorm* and in particular its VHDL `port` declaration, the input is denoted `IN1` and the output is denoted `OUT1`.[51]

94. The design of the *denorm* module is relatively straightforward, as reflected in the VHDL code. The *denorm* module copies or forwards the input's (`IN1`) sign bit to the sign bit of the output (`OUT1`).[52] Similarly, the *denorm* module copies or forwards the exponent bits of the input (`IN1`) to the exponent bits of the output (`OUT1`).[53] For the mantissa, the *denorm* module copies or forwards the fraction bits of the input (`IN1`) to the corresponding mantissa bits of the output (`OUT1`) and adds a leading '1' (*i.e.*, the implied '1') to the most-significant bit of the output mantissa.[54] To accommodate the insertion of the additional bit in the output mantissa, the output of the *denorm* module (`OUT1`) is one bit wider than than the input (`IN1`), and the *denorm* module internally uses fraction signals that are one bit wider for the output mantissa (`f_new`) than the input fraction (`f_old`).[55] When the input floating-point number is '0', which is represented according to IEEE convention by all zero exponent and fraction/mantissa bits, the

[50] *See* Exhibit D at denorm.vhd.
[51] *See* Exhibit D at denorm.vhd:60-70.
[52] *See* Exhibit D at denorm.vhd:77-78, 87, 104, 110.
[53] *See* Exhibit D at denorm.vhd:79-80, 88, 105, 111.
[54] *See* Exhibit D at denorm.vhd:81-82, 89, 92-101, 106-107, 112.
[55] *See* Exhibit D at denorm.vhd:63, 67 (defining `IN1` as a vector with `exp_bits` + `man_bits` + 1 elements, and OUT1 as a vector with `exp_bits` + `man_bits` + 2 elements); Exhibit D at denorm.vhd:81-82 (defining `f_old` as a vector with `man_bits` elements, and `f_new` as a vector with `man_bits` + 1 elements).

denorm module inserts a '0' bit rather than the implied '1' at the most-significant bit of the output mantissa, so that the output remains '0'. To accommodate this scenario, the *denorm* module checks the input exponent bits using a simple *n*-input OR gate that is built from the lower-level hardware module *parameterized_or_gate*.[56] If any of those bits are '1', the OR gate will output a '1', which is used to insert the implied '1' into the most-significant bit of the output mantissa; otherwise the output of '0' from the OR gate will be used to insert a '0' into the most-significant bit of the output mantissa.[57]

b. VFLOAT Exponent Addition and Mantissa Multiplication Module: *fp_mul*

95. As noted above, the parameterized arithmetic module in VFLOAT for performing exponent addition and mantissa multiplication was named *fp_mul*, and its design (*i.e.*, structure and behavior) is expressed in the VHDL source code specified in fp_mul.vhd.[58] The module *fp_mul* accepts as inputs two floating-point numbers and outputs the sum of their exponents and the product of their mantissas. As can be seen in the source code for *fp_mul* and in particular its VHDL `port` declaration (which defines the interfaces for a VHDL entity, *i.e.*, the inputs and outputs to/from a block), those inputs are denoted `OP1` and `OP2`.[59] Similarly, the output is denoted in the `port` declaration as `RESULT`.[60]

96. The module *fp_mul* includes a fixed-point adder (denoted `exponent_adder` in the code), which is built from the lower-level hardware module *parameterized_adder*[61] and sums together the exponents of the two floating-point inputs.[62] The *fp_mul* module also includes a fixed-point subtractor (denoted `bias_subtractor` in the code), which is built from the lower-level hardware module *parameterized_subtractor*[63] and subtracts the exponent bias from

[56] *See* Exhibit D at denorm.vhd:92-101; Exhibit D at parameterized_or_gate.vhd.
[57] *See* Exhibit D at denorm.vhd:88, 92-101, 107, 112.
[58] *See* Exhibit D at fp_mul.vhd.
[59] *See* Exhibit D at fp_mul.vhd:65-69.
[60] *See* Exhibit D at fp_mul.vhd:65-74.
[61] *See* Exhibit D at parameterized_adder.vhd.
[62] *See* Exhibit D at fp_mul.vhd:127-139, 198, 201.
[63] *See* Exhibit D at parameterized_subtractor.

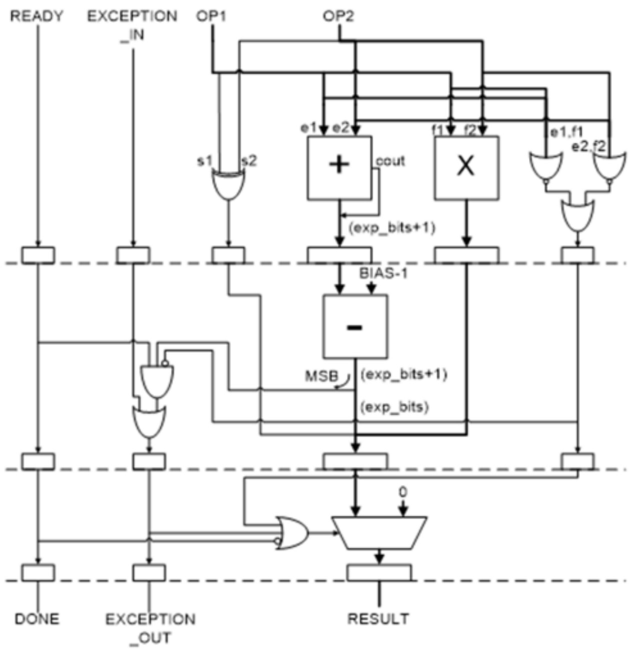
the sum of the input exponents in order to account for the fact that the input exponents are biased.[64] The module *fp_mul* also includes a fixed-point multiplier (denoted `mantissa_multiplier` in the code), which is built from the lower-level hardware module *parameterized_multiplier*[65] and multiplies the mantissas of the two floating-point inputs.[66]

97. The *fp_mul* module also includes a number of registers used for timing purposes.[67] As can be seen in the source code for *fp_mul* and in particular its VHDL `port` declaration, the module also accepts a control signal, denoted `READY` in the code, that instructs it to begin processing data.[68]

98. The *fp_mul* module also receives an exception signal (`EXCEPTION_IN`) and can output its own exception signal (`EXCEPTION_OUT`), as can also be seen in the source code.[69] One circumstance in which the value of `EXCEPTION_OUT` will be set to 1, to signal an exception, is if the output of the *paramterized_subtractor* was either too large or too small to be represented in the exponent field of the output.

99. When implemented in hardware, an *fp_mul* module creates a circuit on the FPGA with the following structure:[70]

[64] *See* Exhibit D at fp_mul.vhd:151-161, 224.
[65] *See* Exhibit D at parameterized_multiplier.vhd.
[66] *See* Exhibit D at fp_mul.vhd:140-150, 199, 202.
[67] *See* Exhibit D at fp_mul.vhd:215-235 (specifying synchronous assignment of certain timing registers).
[68] *See* Exhibit D at fp_mul.vhd:65-77.
[69] *See* Exhibit D at fp_mul.vhd:65-77.
[70] This diagram is taken from the slides for my presentation at HPEC 2002 (which is discussed in more detail below).



100. The `RESULT` output of the *fp_mul* module is designed to accommodate the same number of exponent bits as the inputs but twice the number of mantissa bits as the inputs, because of the mathematical fact that performing the fixed-point multiplication of the mantissas results, at least initially (*i.e.*, before any rounding), in a product mantissa that is wider than the mantissa of the inputs (as noted above, the output mantissa is calculated as the fixed-point products of the input mantissas).[71] Returning the output of *fp_mul* to the same bitwidth as the inputs to the multiplication operation is one function of the *rnd_norm* module, discussed in more detail below.

101. As discussed above, VFLOAT arithmetic modules were designed to accept input exponents that were biased using a biasing scheme similar to that used in the IEEE single-precision format. As discussed above, in such a biasing scheme, a bias is applied to determine the stored exponent value, and the value of that bias that depends on the bitwidth of the exponent. As discussed above, assuming an *n*-bit biased exponent, the bias can be formulaically expressed as: $2^{n-1} - 1$. For example, in the case of a 6-bit biased exponent, the bias is $2^{6-1} - 1 =$

[71] *See* Exhibit D at fp_mul.vhd:74 (defining `RESULT` as `std_logic_vector(exp_bits+(2*man_bits) downto 0)`).

$2^5 - 1 = 32 - 1 = 31$. In the case of the *fp_mul* module, the use of a biasing scheme can be seen, for example, in the use of the `bias_subtractor`, which takes the sum of the input exponents and subtracts one copy of the bias from that sum.[72]

c. VFLOAT Floating-Point Rounding and Normalization Module: *rnd_norm*

102. As noted above, the parameterized module in VFLOAT for performing normalization and rounding on floating-point values was named *rnd_norm*, and its design (*i.e.*, structure and behavior) is expressed in the VHDL source code specified in rnd_norm.vhd.[73] The *rnd_norm* module accepts as input one floating-point number, performs normalization on that number, rounds it, then outputs the result.

103. As can be seen in the source code for *rnd_norm* and in particular its VHDL `port` declaration, the input is denoted `IN1` and the output is denoted `OUT1`.[74] As discussed above, the input `IN1` and the output `OUT1` are parameterized based on the values specified by `exp_bits`, `man_bits_in`, and `man_bits_out`. Specifically, the parameter `man_bits_in` specifies the mantissa bitwidth for the input `IN1`, the parameter `man_bits_out` specifies the mantissa bitwidth for the output `OUT1`, and the parameter `exp_bits` specifies the exponent bitwidth for both the input `IN1` and the output `OUT1`.[75] The reason that different parameters are used to specify the mantissa bitwidths for the input and output of *rnd_norm* is that the rounding operation performed by *rnd_norm* entails a reduction in the mantissa bitwidth. The difference between `man_bits_in` and `man_bits_out` corresponds to the number of bits to be eliminated from the mantissa in the rounding and normalization process. As can also be seen in the source code for *rnd_norm* and in particular its VHDL `port` declaration, the module also accepts a control signal, denoted `READY` in the code, that instructs it to begin processing data.[76]

[72] *See* Exhibit D at fp_mul.vhd:137, 151-161, 224. The exponent bias is computed previously in the code for *fp_mul* based on the bitwidth of the input exponents. *See* Exhibit D at fp_mul.vhd:212-214.
[73] *See* Exhibit D at rnd_norm.vhd.
[74] *See* Exhibit D at rnd_norm.vhd:61-73.
[75] *See* Exhibit D at rnd_norm.vhd:55-60, 64, 70.
[76] *See* Exhibit D at rnd_norm.vhd:61-73.

104. Within the *rnd_norm* module, the sign bit of input `IN1` is simply propagated in pipelined fashion through a series of timing registers to the sign bit of output `OUT1` because normalization and rounding do not affect the sign of the floating-point input.[77]

105. The *rnd_norm* module first performs normalization of the floating-point input `IN1` using the `norm` component, which is built from the lower-level hardware module *normalizer* and performs normalization by shifting the input mantissa bits to normalized form, incrementing or decrementing the exponent accordingly, and dropping the leading integer bit (the implied '1').[78]

106. The *rnd_norm* module then performs rounding to ultimately output a result that has an overall bitwidth of *m* = `exp_bits` + `man_bits_out` + 1.[79] Rounding refers to the well-known process of selecting an approximation of a particular value that fits within a desired degree of precision. For example, it may be undesirable, unnecessary, or impossible to store all the bits of the number 3.14159265359 (which is itself an approximation of the number π, pi). Instead, one might *round* this number to 3.14 and store only that value due to expedience, necessity, or otherwise.

107. The module *rnd_norm* supports two modes (*i.e.*, methods) of rounding: a method commonly known as rounding to zero, and another method commonly referred to as rounding to nearest. In the first mode, rounding to zero, lower-order bits in the mantissa are simply truncated, *i.e.*, deleted, thereby reducing the mantissa bitwidth to the desired size. Truncating lower-order bits from a floating-point value can only have the effect, if any, of reducing the overall magnitude of the number (*i.e.*, bringing it closer to zero), which is why this mode of rounding is commonly referred to as rounding to zero.

108. The second rounding mode, rounding to nearest, operates more or less as its name suggests: the input is rounded to the nearest value representable within the bitwidth of the output

[77] *See* Exhibit D at rnd_norm.vhd:80-83, 101, 106, 156-158.
[78] *See* Exhibit D at normalizer.vhd.
[79] *See* Exhibit D at rnd_norm.vhd:70.

mantissa, with the input rounded to the higher representable value when it is equidistant to the two nearest representable values. The way this is implemented, essentially, is by adding '1' to the input mantissa bit that is one digit to the right of the least-significant bit of the output mantissa bitwidth (*i.e.*, to the most-significant bit that might otherwise be truncated based on the output mantissa bitwidth), then truncate to the desired bitwidth of the output mantissa. (The `ROUND` signal discussed below is used as the '1' that is used in this addition; thus, if `ROUND` is set to 0, which is for the round-to-zero mode of rounding, then the '1' is not added.) As an illustration of how this works, consider a decimal (base 10) input with one integer digit and one fraction digit, and round to nearest is being used to round to the nearest integer. This procedure would add 0.5 to the input, then truncate all digits after the radix (decimal) point. Thus, 4.1 + 0.5 = 4.6, which would be rounded via simple truncation to 4. And 4.6 + 0.5 = 5.1, which would be rounded via simple truncation to 5. For a value that presents a "tie," 4.5 + 0.5 = 5.0, which would be rounded via simple truncation to 5.

109. The rounding mode is a run-time parameter specified by the input signal `ROUND`[80] When `ROUND` is '1', rounding to nearest is used; when `ROUND` is not set (*i.e.*, the signal is '0'), rounding to zero is used. The different rounding modes in *rnd_norm* are implemented using the `rnd_add` component, which is built from the lower-level hardware module *round_add* by feeding the rounding-mode signal `ROUND` into the `rnd_add` component.[81]

C. VFLOAT System Setup and Implementation at Northeastern University

110. As discussed above, VFLOAT is a library of parameterized hardware modules for performing arithmetic operations on variable-precision floating-point numbers using FPGAs. In addition to developing VFLOAT and making it available to others as described below, we also mapped the designs to FPGA hardware in order to validate and further our work; for example, to evaluate how many complete floating point arithmetic operators could be deployed within a particular FPGA.

[80] *See* Exhibit D at rnd_norm.vhd:67, 145, 162.
[81] *See* Exhibit D at round_add.vhd; Exhibit D at rnd_norm.vhd:133-151, 161-162.

111. The specific FPGA hardware we used was the first-generation "WILDSTAR" reconfigurable computing engine, which was commercially available from Annapolis Micro, a seller of computing hardware based in Annapolis, Maryland. The WILDSTAR reconfigurable computing engine was first announced by Annapolis Micro in mid-1999 and had been available commercially for a year or two by the time of our work.[82] As noted above, Annapolis Micro produced subsequent iterations of WILDSTAR reconfigurable computing engine in later years, including, by 2008, the WILDSTAR 5, which featured 3 Xilinx Virtex-5 FPGAs.[83]

112. The particular form factor of the WILDSTAR reconfigurable computing engine that we used in our work was the PCI board. Shown below is an exemplary photo of the WILDSTAR PCI board.[84] The board's PCI bus interface, which can be seen in the below photo, was used to connect it to a host workstation.

113. The WILDSTAR board we used contained 3 Xilinx Virtex XCV1000 FPGAs, each containing the equivalent of just over 1,000,000 (1,124,022) system gates for a total of 6,144 CLBs (with each CLB comprising 2 slices for a total of 12,288 slices per FPGA).[85] The WILDSTAR board also had built-in random access memory that was accessible by the Xilinx

[82] *See* Xcell, Vol. 32 at 40 (Q2 1999), *available at* https://www.xilinx.com/publications/archives/xcell/Xcell32.pdf [LEESER000041 at 80].
[83] *See* https://web.archive.org/web/20081004190813/http://www.annapmicro.com/ws5pci.html [LEESER000120].
[84] This photo is from the slides for my presentation at HPEC 2002 in September 2002, which is discussed below.
[85] *See* Virtex™ 2.5 V Field Programmable Gate Arrays Datasheet at 1 (v1.7, Oct. 1, 1999) [XILINX-GOOG-SUB00000069 at 69]; Virtex™ 2.5 V Field Programmable Gate Arrays Datasheet at 1 (v2.5, April 2, 2001) [XILINX-GOOG-SUB00000522 at 522].

FPGAs. Specifically, the WILDSTAR board had 40 megabytes of SRAM, which was accessible to both the host workstation as well as each of the Xilinx FPGAs.

114. Our WILDSTAR board was installed in a standard Intel-based, x86 host workstation that was located in the RPL at Northeastern University. The workstation's CPU was a commercially available Intel Pentium III processor. The Intel Pentium III processor contained four IEEE standard floating-point multipliers, which were each capable of performing multiplication on IEEE single-precision floating-point values (*i.e.*, floating-point numbers in the 32-bit IEEE single-precision format).[86]

115. Once a bitstream has been generated and downloaded to one of the WILDSTAR's FPGAs, the resulting hardware or circuitry programmed onto the FPGA is activated by a "start" signal sent from the workstation, which was functionality provided by a board support package.

116. As noted above, in addition to the physical WILDSTAR board itself, Annapolis Micro provided software referred to as a board support package (BSP) that ran on the x86 host machine and supported downloading the bitstream from the host, runtime communication with the host (e.g., sending a "start" signal), and communication with on-board memory.

D. Estimating Size of VFLOAT Arithmetic Modules and Mapping Many VFLOAT Modules to FPGAs on the WILDSTAR Reconfigurable Computing Engine

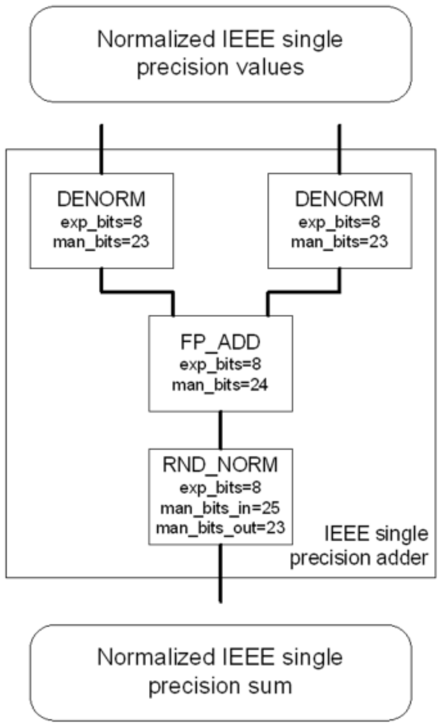
117. One area of interest in our development and implementation of VFLOAT, which stemmed from the goal of allowing greater parallelism and more efficient use of hardware resources through flexible control over signal bitwidths, was to determine how many complete VFLOAT arithmetic operators could in practice be mapped to FPGA hardware with a given number of resources. Because the size of a complete VFLOAT arithmetic operator (and thus how many could "fit" within a given FPGA) depends on the bitwidth of the floating-point format it takes as inputs and produces as output, we explored this question using a variety of different

[86] *See* Microprocessor Report (Vol. 13, No. 3, Mar. 8, 1999), *available at* https://docencia.ac.upc.edu/ETSETB/SEGPAR/microprocessors/pentium3%20(mpr).pdf [LEESER000130].

floating-point number formats supported by VFLOAT, ranging from total bitwidths of as few as 8 bits up to as many as 32 bits, including the 32-bit IEEE single-precision format. One reason for this experiment is that we desired to perform, and show to others, an "apples-to-apples" comparison of how many complete floating-point operators would fit on an FPGA using the IEEE single-precision format versus other formats with lower bitwidths.

118. Our investigation of these issues was based on assembling, implementing, and evaluating arithmetic operators that perform a complete floating-point operation, meaning operators that included both (i) denormalization (*denorm*) modules on the input side, and (ii) a rounding and normalization (*rnd_norm*) module on the output side, which I sometimes refer to in this section as "complete" arithmetic operators. For example, when determining how many complete VFLOAT floating-point addition operators using a given floating-point format would fit on an FPGA, our assessment was based on assembling and synthesizing the complete operator using two instances of an appropriately parameterized VFLOAT denormalization module *denorm* (one for each of the input operands), one instance of an appropriately parameterized VFLOAT floating-point addition module *fp_add* (to perform floating-point addition), and one instance of an appropriately parameterized VFLOAT rounding and normalization module *rnd_norm* (to perform rounding and normalization on the output of *fp_add*). For example, as part of our analysis, we assembled and evaluated complete VFLOAT floating-point addition operators for the IEEE single-precision floating-point format (1 sign bit, 8 exponent bits, and 23 mantissa bits) using two *denorm* modules, one *fp_add* module, and one *rnd_norm* module. This is illustrated in the following diagram,[87] which shows how we built an IEEE single-precision adder using VFLOAT modules as just described:

[87] This diagram is taken from the slides for my presentation at HPEC 2002 (which is discussed in more detail below).


Normalized IEEE single precision values
DENORM exp_bits=8 man_bits=23
DENORM exp_bits=8 man_bits=23
FP_ADD exp_bits=8 man_bits=24
RND_NORM exp_bits=8 man_bits_in=25 man_bits_out=23
IEEE single precision adder
Normalized IEEE single precision sum

119. With respect to the complete VFLOAT floating-point addition operator for the IEEE single-precision floating-point format, we determined that 31 such operators would fit on just one of the three Xilinx Virtex XCV1000 FPGAs present on the WILDSTAR reconfigurable computing engine, and that a single complete VFLOAT IEEE single-precision addition operator used 305 slices of the FPGA.[88]

120. We also evaluated this issue for other floating-point formats besides IEEE single-precision, including the number of such operators that could fit on a single Xilinx Virtex XCV1000 FPGA. We synthesized complete addition and multiplication operators for a variety of floating-point formats with total bitwidths of 8, 12, 16, 24, and 32 bits to determine how many would fit on a single Xilinx Virtex XCV1000 FPGA. These included, for example, a 16-bit

[88] This is corroborated, for example, by the slides for my presentation at HPEC 2002 (discussed in more detail below), at which I discussed how our work with VFLOAT demonstrated that 31 adders for the IEEE single-precision format would fit on a Xilinx VirtexXCV1000 FPGA. *See* Exhibit C at 5, 19. It is also corroborated by the abstract that I co-authored with Mr. Belanović for presentation at FPL 2002, which is discussed further below. *See* P. Belanović & M. Leeser, *A Library of Parameterized Floating Point Modules and Their Use* at 663-64 (Springer-Verlag 2002) ("The total design mapped to a Xilinx Virtex 1000 takes up 305 slices, or just under 2.5% of the chip.") [GOOG-SING-00020145 at 20151-52].

format with 6 exponent bits and 9 fraction bits, which we denominated the "C2" format; a 16-bit format with 5 exponent bits and 10 fraction bits, which we denominated the "C1" format; and a 16-bit format with 4 exponent bits and 11 fraction bits, which we denominated the "C0" format.[89] For all of the formats we considered, we implemented the complete VFLOAT floating-point addition and multiplication operators, as described above (*i.e.*, including the denormalization and rounding/normalization modules), in FPGA hardware and determined the maximum number of such operators that could in practice be mapped to a Xilinx Virtex XCV1000 FPGA. In order to ensure that our determinations regarding the maximum number of complete VFLOAT floating-point addition and multiplication operators that could fit on a Xilinx Virtex XCV1000 FPGA were realistic, we assumed that a certain percentage of resources on the FPGA would need to be reserved for routing overhead, and thus would be unavailable. Specifically, we conservatively assumed that approximately 15% of FPGA resources would need to be reserved for routing overhead, leaving only about 85% of the FPGA for floating-point operators.

121. For the C2 format (16-bit floating-point format with 6 exponent bits and 9 fraction bits), for example, we determined that 81 complete floating-point addition operators would fit on a single Xilinx Virtex XCV1000 FPGA, and 61 complete floating-point multiplication operators would fit on a single Xilinx Virtex XCV1000 FPGA. As a further example, for the C1 format (16-bit floating-point format with 5 exponent bits and 10 fraction bits), we determined that 65 complete floating-point addition operators would fit on a single Xilinx Virtex XCV1000 FPGA, and 51 complete floating-point multiplication operators would fit on a single Xilinx Virtex XCV1000 FPGA. As an additional example, for the C0 format (16-bit floating-point format with 4 exponent bits and 11 fraction bits), we determined that 76

[89] This is corroborated, for example, by Mr. Belanović's written thesis. *See* P. Belanović, *Library of Parameterized Hardware Modules for Floating-Point Arithmetic with An Example Application* at 46-47 (Northeastern Univ. 2002) (listing multiple 16-bit floating-point formats used in synthesis work with VFLOAT, including "C0" format with 4-bit exponent and 11-bit fraction, "C1" format with 5-bit exponent and 10-bit fraction, and "C2" format with 6-bit exponent and 9-bit fraction) [GOOG-SING-00020062 at 20107-108].

complete floating-point addition operators would fit on a single Xilinx Virtex XCV1000 FPGA, and 44 complete floating-point multiplication operators would fit on a single Xilinx Virtex XCV1000 FPGA.[90]

122. To confirm and validate our conclusions, for each floating-point format that we considered, we actually synthesized the maximum number of complete VFLOAT floating-point multiplication and addition operators that we determined would fit on a single Xilinx Virtex XCV1000 and mapped that number of operators to the FPGA. For example, for each floating-point number format under consideration, we synthesized the maximum number of complete VFLOAT floating-point multiplication operators that we determined would fit on a single Xilinx Virtex XCV1000, and mapped that number of complete operators to one of the Xilinx Virtex XCV1000 FPGAs on our WILDSTAR board by generating a bitstream using commercially available FPGA software tools (such as Xilinx Alliance tools and Synplicity Pro). We then loaded the bitstream onto the Xilinx Virtex XCV1000 (using the BSP from Annapolis Micro), which resulted in circuits in the FPGA hardware embodying that number of complete VFLOAT floating-point multiplication operators for the given floating-point format. The same is true for complete VFLOAT floating-point addition operators.

123. For example, relative to the "C2" format discussed above, we synthesized a design that contained 61 complete VFLOAT floating-point multiplication operators—which, as explained above, we determined was the maximum number that would fit on one of the three Xilinx Virtex XCV1000 FPGAs in our WILDSTAR board—and loaded the corresponding bitstream onto a Xilinx Virtex XCV1000 to implement that many complete VFLOAT floating-point multiplication operators in hardware. Similarly, we also synthesized a design that contained 81 complete VFLOAT floating-point addition operators using the "C2" format—which, as explained above, we determined was the maximum number that would fit on one of the

[90] These conclusions regarding the C0, C1, and C2 formats are corroborated, for example, by Mr. Belanović's written thesis. *See* P. Belanović, *Library of Parameterized Hardware Modules for Floating-Point Arithmetic with An Example Application* at 47 (Northeastern Univ. 2002) [GOOG-SING-00020062 at 20107].

three Xilinx Virtex XCV1000 FPGAs in our WILDSTAR board—and loaded the corresponding bitstream onto a Xilinx Virtex XCV1000 to implement that many complete VFLOAT floating-point addition operators in hardware.

E. Public Disclosures Relating to VFLOAT and Our System Setup

124. The original code comprising VFLOAT, along with documentation describing the purpose and functionality of the library, was made publicly available via a dedicated webpage on the website for Northeastern University's Department of Electrical and Computer Engineering around the time the initial version was completed (*i.e.*, in approximately 2002), and has remained publicly available since then. For example, the Internet Archive's Wayback Machine maintains a true and correct copy of the website for VFLOAT as it existed in early March 2003,[91] a copy of which is attached to my report as **Exhibit E**. The current webpage for VFLOAT is https://coe.northeastern.edu/Research/rcl/projects/floatingpoint/index.html [LEESER000146] and contains a link to the original code developed in 2002. That code can still be downloaded and, as noted earlier, is attached in full to my report as **Exhibit D**.

1. Mr. Belanović's May 2002 Oral Thesis Defense

125. In addition to making the code for VFLOAT publicly available via a dedicated webpage as described above, I and others presented VFLOAT and our related work through various presentations, conferences, workshops, meetings, and the like. For example, in May 2002, Mr. Belanović gave an oral presentation in connection with defending his thesis, which was titled "Library of Parameterized Modules for Floating-Point Arithmetic with An Example Application." Specifically, Mr. Belanović's thesis defense took place in a presentation at 10:00 a.m. on Wednesday, May 8, 2002, in Room 206 of the Egan Research Center, which is on the Northeastern University campus at 360 Huntington Ave., Boston, Massachusetts.

[91] *See* https://web.archive.org/web/20030313114351/http://www.ece.neu.edu:80/groups/rpl/projects/floatingpoint/index.html.

126. As is typical for a thesis defense, Mr. Belanović's presentation to the thesis committee was public, and he was later awarded his M.S. degree in Electrical Engineering from the Northeastern University Department of Electrical and Computer Engineering on the basis of the presentation and underlying work. Prior to Mr. Belanović's oral presentation of his thesis, an email was distributed to the faculty and students of the Department of Electrical Engineering announcing his thesis defense presentation and inviting any interested members of the community to attend. The substance of that email announcement, which I saved by way of an email forwarded to myself in approximately 2005, is attached as **Exhibit F**.

127. As noted, Mr. Belanović's thesis presentation was public, meaning that any members of the Northeastern University community or the general public were welcome; there were no restrictions on who was permitted to attend Mr. Belanović's thesis defense, nor were attendees subject to any restrictions on subsequent disclosure or discussion of the presentation. As Mr. Belanović's advisor and a member of his thesis committee, I attended Mr. Belanović's thesis defense in May 2002, as did other members of the committee including Dana Brooks and Waleed Meleis.[92] Other graduate students I supervised at the time attended the thesis defense as well, including Wang Chen, Haiqian Yu, and Heather Quinn, as it was my practice to invite all my graduate students and suggest that they attend the thesis defense of any graduate student whom I supervised. Mr. Belanović's oral thesis defense presentation would have lasted approximately 1 hour—divided between approximately 45 minutes of presentation by Mr. Belanović followed by approximately 15 minutes for a question-and-answer session—which was the typical amount of time allotted for a thesis defense at the Department of Electrical and Computer Engineering.

128. During his thesis defense, Mr. Belanović orally presented on a number of aspects of VFLOAT and our work related to VFLOAT, including displaying and discussing results tables, such as Table 2.2 and Figures 2.10 and 2.11 from his written thesis, relating to the

[92] The members of Dr. Belanović's thesis committee are identified in the first few pages of his written thesis. *See* GOOG-SING-00020062 at 20063-64.

conclusions we reached regarding the maximum number of complete VFLOAT floating-point arithmetic operators that would fit on a single one of the three Xilinx Virtex XCV1000 FPGAs on the WILDSTAR board that we used.

2. Other Disclosures at Northeastern University

129. Other members of the Northeastern University community also learned of and became familiar with VFLOAT during the time it was being developed and in subsequent years after it was made publicly available, including for example the graduate students who contributed to later iterations or revisions of VFLOAT over the years (*e.g.*, Haiqian Yu in 2003 and Xiaojun Wang in 2008) as well as several other graduate students whom I supervised, including Shawn Miller, Joshua Noseworthy, Albert A. Conti III, and Ben Cordes. The RPL itself, where our physical workstation described above was located, was a shared space in which, at any given time, at least a dozen and possibly as many as 20 other students and faculty members within the Department of Electrical and Computer Engineering routinely worked and to which they had unrestricted access. In addition, I specifically recall demonstrating VFLOAT and our physical workstation that included the WILDSTAR reconfigurable computing engine in approximately 2002 to Laurie Smith King, a Professor of Computer Science at Holy Cross University, who spent a sabbatical at Northeastern University during that timeframe.

3. HPEC 2002

130. In addition, during my secondary appointment as a Visiting Scientist at MIT's Lincoln Laboratories from June to December 2002 (discussed above), I attended and presented at the Sixth Annual Workshop on High Performance Embedded Computing (HPEC 2002), which was held at the MIT Lincoln Laboratory September 24-26, 2002. HPEC was intended as a forum where researchers from academia, industry, and government can discuss techniques, approaches, and ongoing developments relevant to high-performance real-time computing. A copy of the agenda for HPEC 2002 is attached as **Exhibit G**. In advance of the workshop, I authored (along with Mr. Belanović) a 2-page abstract with a summary of our proposed presentation (the "HPEC

2002 Abstract"), which was titled "A Library of Parameterized Hardware Modules for Floating-Point Arithmetic and Their Use" and was submitted to (and accepted by) the workshop organizers for presentation at HPEC 2002. The HPEC 2002 Abstract is attached as **Exhibit H**.

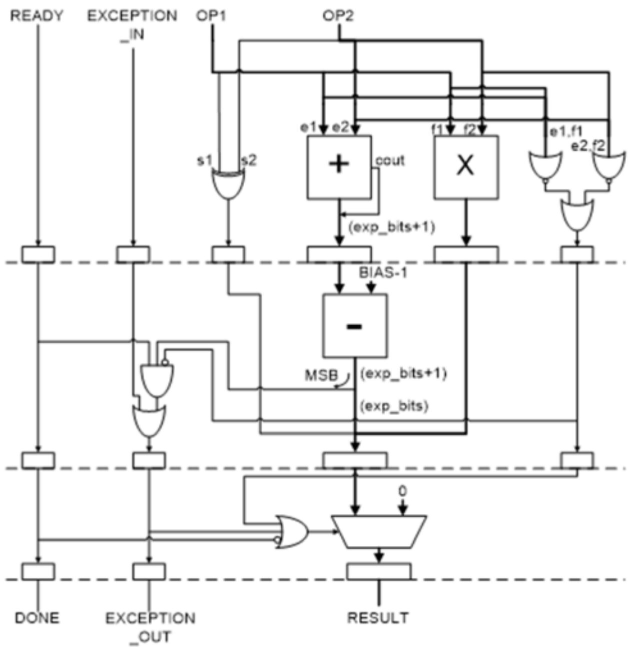
131. As reflected in the agenda for HPEC 2002, on the first day of the workshop, Tuesday, September 24, 2002, I presented various aspects of our work related to VFLOAT in a session entitled "A Library of Parameterized Hardware Modules for Floating-Point Arithmetic and Its Use."[93] Approximately 100 individuals attended HPEC 2002 overall, and approximately 30-40 individuals attended the session in which I presented various aspects of VFLOAT. The slides accompanying my oral presentation at HPEC 2002 are attached as **Exhibit C**. As reflected in those slides, I presented on a number of different aspects of VFLOAT, our related work, and applications thereof.

132. As reflected in my HPEC 2002 slides, I presented and discussed certain hardware details of the workstation we used in the RPL at Northeastern University for implementing VFLOAT.[94] For example, I explained that our system setup used a WILDSTAR reconfigurable computing engine from Annapolis Micro and that the WILDSTAR board had three Xilinx Virtex XCV1000 FPGAs.[95] I also discussed that we had developed VFLOAT in VHDL and mapped VFLOAT arithmetic modules to the Xilinx FPGAs.[96]

133. Similarly, during my presentation at HPEC 2002, I discussed the various modules that comprised VFLOAT, including specifically discussing and explaining the design (*i.e.*, structure and behavior) of the exponent addition and mantissa multiplication module (*fp_mul*), the denormalization module (*denorm*), and the normalization and rounding module (*rnd_norm*).[97] For example, I disclosed and explained that, as part of a complete floating-point multiplication operation, *fp_mul* was the VFLOAT arithmetic module that performed exponent

[93] *See* Exhibit G (HPEC 2002 Agenda).
[94] *See* Exhibit C at 17 (HPEC 2002 Slides).
[95] *See* Exhibit C at 17 (HPEC 2002 Slides).
[96] *See* Exhibit C at 17 (HPEC 2002 Slides).
[97] *See* Exhibit C at 8, 11-16 (HPEC 2002 Slides).

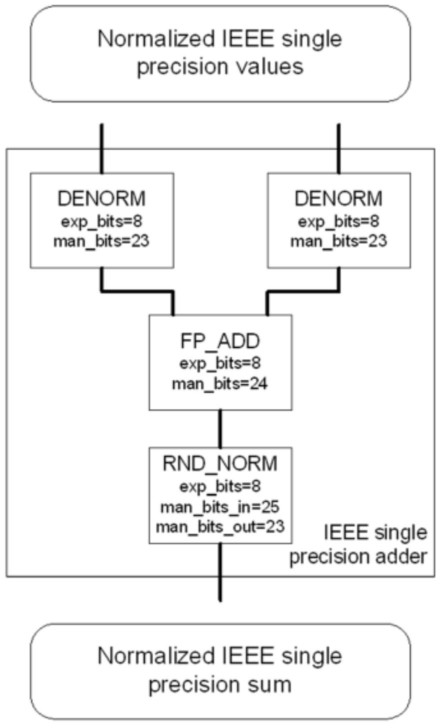
addition and mantissa multiplication, *denorm* was the VFLOAT module that "insert[s] [the] implied digit" into a normalized floating-point representation of a number, and *rnd_norm* was the VFLOAT module that "[r]eturns [its] input to normalized format."[98] In addition, I specifically discussed and disclosed the structure of the circuits that result from implementing each of the various VFLOAT floating-point arithmetic modules in hardware. For example, I discussed and disclosed to attendees the structure of the circuit that results from implementing the module *fp_mul* in hardware, using the following diagram:[99]



134. I also discussed how to assemble various VFLOAT modules to create a complete circuit for performing floating-point arithmetic operations such as addition and multiplication. For example, I specifically explained how to assemble a complete floating-point operator in hardware using VFLOAT by combining and connecting two *denorm* modules, one *fp_add* module, and one *rnd_norm* module.[100] In addition, for example, I discussed and disclosed how the *rnd_norm* module, due to its function of performing normalization and rounding, was

[98] *See* Exhibit C at 11-12, 14 (HPEC 2002 Slides).
[99] *See* Exhibit C at 14 (HPEC 2002 Slides).
[100] *See* Exhibit C at 10 (HPEC 2002 Slides).

"[d]esigned to follow arithmetic operation(s)."[101] This discussion and disclosure is also illustrated, for example, in the following diagram that I included in the slides for my HPEC 2002 presentation:[102]



135. During my presentation at HPEC 2002, I further discussed and disclosed our synthesis results and conclusions regarding the number of complete floating-point arithmetic operators that would fit on the hardware we used. This was an important part of my presentation because one of the principal motivations I identified in the presentation was to exploit parallelism to accelerate applications.[103] In particular, I compared the number of circuits embodying complete floating-point arithmetic operators of relatively lower precision that could be deployed in parallel on an FPGA to the number of such circuits that could be deployed in parallel if using IEEE single-precision formats.[104] For example, I disclosed and discussed our results showing that 85 complete floating-point multipliers using a 12-bit format could fit on a

[101] *See* Exhibit C at 12 (HPEC 2002 Slides).
[102] *See* Exhibit C at 10 (HPEC 2002 Slides).
[103] *See* Exhibit C at 3, 18-19 (HPEC 2002 Slides).
[104] *See* Exhibit C at 5, 18-19 (HPEC 2002 Slides).

single Xilinx Virtex XCV1000 FPGA whereas only 13 complete floating-point multipliers could fit on the same FPGA if using the IEEE single-precision format (with 32 bits).[105]

136. I also specifically discussed and disclosed during my HPEC 2002 presentation our conclusions regarding the median number of complete floating-point arithmetic operators that would fit on a single Xilinx Virtex XCV1000 FPGA, and further discussed and disclosed examples of specific floating-point number formats for each bitwidth.[106] For example, I specifically discussed and disclosed that at least 50 complete floating-point multipliers built from VFLOAT modules and using a 16-bit floating-point format would fit on a single Xilinx Virtex XCV1000 FPGA.[107]

4. Disclosures to / at Los Alamos National Laboratory

137. Further public disclosures related to VFLOAT took place in the context of meetings with, and presentations to, officials at LANL. As discussed above, VFLOAT arose out of and became the deliverable for our work for LANL under the sub-contract discussed above (*i.e.*, "Acceleration of Scene Classification and Spectral Unmixing with Reconfigurable Computing," as identified in my CV). In addition to actually delivering VFLOAT to LANL as part of the work described above, I personally traveled to New Mexico on several occasions in the 2002 timeframe to present our work to LANL, including presenting and describing various custom floating-point number formats that could be used with VFLOAT as well as our synthesis results such as the number of arithmetic cores that could be implemented within a single FPGA unit using various custom floating-point number formats. During those presentations to officials at LANL, we presented on and discussed specific floating-point number formats that may be suitable for use with VFLOAT, including formats with bitwidths smaller than IEEE standard formats, such as a format with a 6-bit biased exponent and 9-bit fraction field. Mr. Belanović also traveled with me to LANL on at least one occasion. During those meetings, we presented

[105] *See* Exhibit C at 5 (HPEC 2002 Slides).
[106] *See* Exhibit C at 18-19 (HPEC 2002 Slides).
[107] *See* Exhibit C at 19 (HPEC 2002 Slides).

our work to officials in the Space and Remote Sensing Sciences Group at LANL, and our presentations to and discussions with them were neither classified nor covered by any type of non-disclosure agreement or obligation.

5. Other Disclosures

138. In September 2002, Mr. Belanović orally presented an overview of VFLOAT and our application of VFLOAT at the 12th International Conference on Field Programmable Logic and Application (FPL 2002). Specifically, Mr. Belanović presented an overview of VFLOAT as well as our particular application of VFLOAT during an afternoon session on Tuesday, September 3, 2002, that was focused on FPGA-based arithmetic.[108] In advance of the conference, Mr. Belanović and I co-authored a paper (the "FPL 2002 Abstract") summarizing the proposed presentation, which summary was titled "A Library of Parameterized Floating Point Modules and Their Use" and was submitted to (and accepted by) the conference organizers for presentation at FPL 2002.[109] The FPL 2002 Abstract was published in the conference proceedings for FPL 2002.[110]

139. At the 7th annual Military and Aerospace Programmable Logic Devices International Conference (MAPLD '04), which was held in September 2004 in Washington, D.C., Xiaojun Wang and I presented an overview of our work related to VFLOAT, including the use of VFLOAT on then-current state of the art FPGAs (specifically, the Xilinx Virtex II XC2V3000 FPGA on a subsequent generation of WILDSTAR hardware from Annapolis Micro), the specific application of VFLOAT to K-means clustering for multispectral satellite images, and our continued work with VFLOAT since it was originally developed. For example, we discussed and disclosed to attendees the various arithmetic modules comprising VFLOAT, including *fp_mul*, *denorm*, and *rnd_norm*. We also described and disclosed our then-recent addition of

[108] *See* https://www.lirmm.fr/fpl02/Prog.html [LEESER000137].
[109] I understand that a copy of the FPL 2002 Abstract was produced in this litigation at GOOG-SING-00020145.
[110] Specifically, the FPL 2002 Abstract was published by Springer-Verlag as part of its Lecture Notes in Computer Science (LNCS) series. *See* https://link.springer.com/chapter/10.1007/3-540-46117-5_68 [LEESER000151].

modules for performing floating-point division (*fp_div*) and floating-point square root (*fp_sqrt*) to VFLOAT. Approximately 100 people attended the MAPLD '04 conference. Our presentation slides for MAPLD '04 are attached to my report as **Exhibit I**. In advance of the conference, Ms. Wang, Haiqian Yu, and I co-authored a paper (the "MAPLD '04 Abstract") summarizing our proposed presentation, which was titled "A Parameterized Floating-Point Library Applied to Multispectral Image Clustering" and was submitted to (and accepted by) the conference organizers for presentation at MAPLD '04. A copy of the MAPLD '04 Abstract is attached to my report as **Exhibit J**.

VII. VALIDITY ANALYSIS OF THE ASSERTED CLAIMS

140. As explained above, I understand that the Asserted Claims in this case are claim 53 of the '273 patent and claim 7 of the '156 patent.

141. Below, I detail my opinions regarding whether the Asserted Claims are invalid as anticipated and/or obvious.

A. '273 Patent, Claim 53

142. Claim 53 of the '273 patent depends from claim 43, which in turn depends from independent claim 36. Including the limitations from claims 36 and 43, claim 53 reads:

> **36.** A device:
>
> comprising at least one first low precision high-dynamic range (LPHDR) execution unit adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,
>
> wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;
>
> wherein the number of LPHDR execution units in the device exceeds the non-negative integer number of execution units in the device adapted to